CS 61B
Spring 2017

Graph Algorithms

Discussion 12: April 12, 2017

1 Graph Algorithms

Traversal Visit all the nodes in the graph.

- Depth-first traversal (preorder and postorder)
- Level-order traversal

Search Given a start node, find a goal state.

- Depth-first search
- Iterative-deepening depth-first search
- Breadth-first search

***Single Pair* Shortest Path** Given a start node, find the shortest path to a goal node.

- *Uniform cost search*
- *Greedy search*
- A* search

***Single Source* Shortest Path** Given a start node, find the shortest paths to all other nodes.

- Dijkstra's algorithm

Minimum Spanning Tree A *spanning tree*, or cycle-free subset of edges connecting all the nodes, with the minimum possible total edge weight.

- Prim's algorithm
- Kruskal's algorithm

2 Minimum Spanning Trees

2.1 Give a description of how Prim's algorithm works.

Starting from any arbitrary source, repeatedly add the shortest edge that connects some node in the tree to some node outside the tree.

Another way of thinking about Prim's algorithm is that it is basically just Dijkstra's algorithm, but where we consider node in order of the distance from the *entire tree*, rather than the distance from the start.

2.2 Give a description of Kruskal's algorithm.

Sort the edges in the graph from least weight to greatest weight and then for every edge, add it to the minimum spanning tree *only if it does not introduce a cycle*.

```
function GRAPH-SEARCH($start$)
    $seen \leftarrow$ an empty set
    $fringe \leftarrow$ java.util.Queue interface
    ADD($start$, $fringe$)
    while $fringe$ is not empty do
        $node \leftarrow$ REMOVE($fringe$)
        if $node$ is the goal then return $node$
        if $node$ is not in $seen$ then
            ADD($node$, $seen$)
            for $child$ in NEIGHBORS($node$) do
                ADD($child$, $fringe$)
    return failure
```

3 Dijkstra's Algorithm

3.1 What fringe do we use in Dijkstra's algorithm and what does it keep track of?

Dijkstra's uses a min-priority queue ordered on the *distance from the start*.

3.2 Assuming the runtime for `changePriority` is in $O(f(N))$ and `removeMin` is in $O(g(N))$ where N is the size of the priority queue, give the runtime of Dijkstra's algorithm on the simple graph $G = (V, E)$.

$O(|E| \cdot f(|V|) + |V| \cdot g(|V|))$

3.3 Give the runtime bound for Dijkstra's assuming the priority queue is implemented using an unordered array, ordered linked list, and a binary min-heap.

$O(|E| \cdot 1 + |V| \cdot |V|)$ for an unordered array if the array indices represent integer vertices and the values in the array represent the priority. $O(|E| \cdot |V| + |V| \cdot 1)$ for an ordered linked list where the greatest expense is in maintaining the order when calling `changePriority`. $O(|E| \cdot \log |V| + |V| \cdot \log |V|)$ for a binary min-heap.

Note that on simple graphs, $|E| \in O(|V|^2)$.

4 A* Search

4.1 What fringe do we use in A* search and what does it keep track of?

A* search uses a min-priority queue ordered on the sum of the *distance from the start (backward cost)* and the *heuristic estimate of distance to a goal (forward cost)*.

4.2 What is a *heuristic*? What is its correctness conditions?

A *heuristic* is a best-guess estimate. In the case of A* search, we use the heuristic to estimate the distance left to travel to the goal. For the purposes of our class, a heuristic is correct if it is *admissible*, or never overestimates the distance to the goal.